

jc777 U.S. PTO
02/24/00

09/511735
jc564 U.S. PTO
02/24/00

UTILITY PATENT APPLICATION TRANSMITTAL

Submit an original and a duplicate for fee processing
(Only for new nonprovisional applications under 37 CFR 1.53(b))

ADDRESS TO:

Assistant Commissioner for Patents
Box Patent Application
Washington, D.C. 20231

Attorney Docket No. 99,447

First Named Inventor Michael S. Borella

Express Mail No. EL028734015US

Total Pages 62

APPLICATION ELEMENTS

1. ☒ Transmittal Form with Fee

2. ☒ Specification (including claims and abstract) [Total Pages 45]

3. ☒ Drawings [Total Sheets 5]

4. ☒ Oath or Declaration [Total Pages 3]

a. ☒ Newly executed

b. ☐ Copy from prior application

[Note Boxes 5 and 17 below]

i. ☐ Deletion of Inventor(s) Signed statement attached deleting inventor(s) named in the prior application

5. ☐ Incorporation by Reference: The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.

6. ☐ Microfiche Computer Program

7. ☐ Nucleotide and/or Amino Acid Sequence Submission

a. ☐ Computer Readable Copy

b. ☐ Paper Copy

c. ☐ Statement verifying above copies

ACCOMPANYING APPLICATION PARTS

8. ☒ Assignment Papers

9. ☒ Power of Attorney

10. ☐ English Translation Document (if applicable)

11. ☐ Information Disclosure Statement (IDS)

☐ PTO-1449 Form

☐ Copies of IDS Citations

12. ☐ Preliminary Amendment

13. ☒ Return Receipt Postcard (Should be specifically itemized)

14. ☐ Small Entity Statement(s)

☐ Enclosed

☐ Statement filed in prior application; status still proper and desired

15. ☐ Certified Copy of Priority Document(s)

16. ☒ Other: Certificate of Express Mail

17. If a CONTINUING APPLICATION, check appropriate box and supply the requisite information:

☐ Continuation ☐ Divisional ☐ Continuation-in-part of prior application Serial No.

APPLICATION FEES				
BASIC FEE				\$690.00
CLAIMS	NUMBER FILED	NUMBER EXTRA	RATE	
Total Claims	39 -20=	19	x \$18.00	\$342.00
Independent Claims	8 - 3=	5	x \$78.00	\$390.00
<input type="checkbox"/> Multiple Dependent Claims(s) if applicable			+\$270.00	\$0.00
Total of above calculations =				\$1,422.00
Reduction by 50% for filing by small entity =				\$()
<input checked="" type="checkbox"/> Assignment fee if applicable			+\$40.00	\$40.00
TOTAL =				\$1,462.00

UTILITY PATENT APPLICATION TRANSMITTAL

Attorney Docket No. 99,447

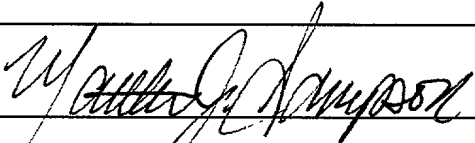
18. ☐ Please charge my Deposit Account No. 13-2490 in the amount of \$.19. ☒ A check in the amount of \$1,462.00 is enclosed.

20. The Commissioner is hereby authorized to credit overpayments or charge any additional fees of the following types to Deposit Account No. 13-2490:

- a. ☒ Fees required under 37 CFR 1.16.
b. ☒ Fees required under 37 CFR 1.17.
c. ☐ Fees required under 37 CFR 1.18.

21. ☐ The Commissioner is hereby generally authorized under 37 CFR 1.136(a)(3) to treat any future reply in this or any related application filed pursuant to 37 CFR 1.53 requiring an extension of time as incorporating a request therefor, and the Commissioner is hereby specifically authorized to charge Deposit Account No. 13-2490 for any fee that may be due in connection with such a request for an extension of time.

22. CORRESPONDENCE ADDRESS

Name	McDonnell Boehnen Hulbert & Berghoff
Address	32 nd Floor, 300 South Wacker Drive
City, State, Zip	Chicago, Illinois 60606
23. SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT REQUIRED	
Name	Matthew J. Sampson
Signature	
Date	February 24, 2000

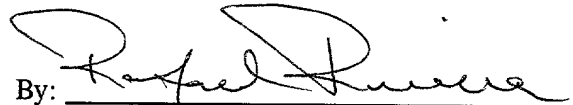
UTILITY (Rev. 11/18/97)

CERTIFICATE OF MAILING BY "EXPRESS MAIL"
(NEW PATENT APPLICATION)
Attorney Docket No.:99,447

Express Mail No. EL028734015US

Deposited February 24, 2000

I hereby certify that the attached correspondence, identified below, is being deposited with the United States Postal Service as "Express Mail Post Office to Addressee" under 37 CFR § 1.10 on the date indicated above and is addressed to the Asst. Commissioner for Patents, Box Patent Application, Washington, DC 20231.

By: 
(person actually depositing)

Patent Application of: Michael S. Borella and Nurettin B. Beser

Title: Method And Application Programming Interface For Assigning Multiple Network Addresses

☒ Patent Application (45 pages, including claims)

☒ Drawings (5 sheets)

☒ Return Postcard

☒ Utility Patent Application Transmittal in Duplicate

☒ Check

☒ Declaration and Power of Attorney

☒ Assignment Recordal Cover Sheet and Assignment

☐ Preliminary Amendment

☐ Other

APPLICATION FOR A UNITED STATES PATENT

UNITED STATES PATENT AND TRADEMARK OFFICE

(CASE No. 99,447)

Title: **METHOD AND APPLICATION PROGRAMMING INTERFACE FOR
ASSIGNING MULTIPLE NETWORK ADDRESSES**

5

10 Inventors: Michael S. Borella, a citizen of the United States of America, and a resident of
Naperville, Illinois; and

Nurettin B. Beser, a citizen of Turkey, and a resident of Evanston, Illinois.

15

20 Assignee: 3Com Corporation
5400 Bayfront Plaza
Santa Clara, CA 95052

FIELD OF INVENTION

The present invention relates to communications in data networks. More specifically, it relates to a method and an application programming interface for assigning multiple network addresses.

BACKGROUND OF THE INVENTION

The Internet Protocol ("IP") and Transmission Control Protocol ("TCP") are rapidly becoming the lingua franca of modern data networks. Currently, host protocol stacks allow a single host to support multiple physical and logical data-link interfaces.

These interfaces may be either software or a combination of hardware and software. A data-link interface is typically a device that permits a host to communicate with another entity. Data-link interfaces may be active for as long as the host is running, or they may activate and deactivate dynamically. An example of a data-link interface is an Ethernet interface which consists of the Ethernet card along with a device driver. Typically the Ethernet interface becomes active upon system boot and remains active until the host computer is shut off.

Data-link interfaces typically represent an IP address that is bound to a data-link device. Communication via these interfaces occurs through device-independent calls to a socket application programming interface ("API"). As is known in the art, a socket is an endpoint, in a host computer's protocol stack, for communicating over a data network. The socket API provides the capability for application programs and their processes to access communications protocols automatically. The socket API exists logically above a transport layer for the TCP/IP

stack, but below an application layer. The socket API, which is well known to those skilled in the art, is discussed in UNIX on-line manuals as well as many textbooks.

At present, all processes typically bind to one common IP address. When a process initiates an IP communication, the protocol stack binds the common IP address to the process. In this sense, all processes typically communicate over the same IP interface. Data traffic to or from one application is typically distinguished from data traffic to or from another application by a transport-layer parameter such as a TCP or User Datagram Protocol ("UDP") port. For some processes, however, having one common IP address may be restrictive.

Currently, protocol stacks may support multiple IP interfaces in a limited sense. A host with more than one physical interface may require more than one IP address. For example, a host computer may communicate using IP packets over a Point-to-point Protocol ("PPP") connection via a modem while simultaneously communicating using IP packets over an Ethernet connection. In this case, the Ethernet interface and the PPP interface would be associated with separate IP addresses. The processes on the host computer, however, have these separate IP addresses in common. Any process that communicates over the modem does so using the common IP address for the PPP connection. Similarly, any process that communicates over the Ethernet connection does so using the common IP address for the Ethernet connection.

Future communication systems, however, may require that a protocol stack is capable of supporting multiple IP addresses in a broader sense: different processes on the same host are associated with different IP addresses on the same physical interface. Processes may request a new IP interface with a new IP address rather than share a single IP address with all other

processes. Instead of distinguishing traffic to and from processes by port numbers, the traffic may be distinguished by IP address.

It is therefore desirable to provide a method for binding multiple IP addresses to the same process. Multiple processes on the same host may then be assigned different IP addresses that are distinguishable at an application layer.

5

SUMMARY OF THE INVENTION

In accordance with preferred embodiments of the present invention, methods for assigning multiple network addresses are provided. One aspect of the invention includes a method for using multiple network addresses for interprocess communication through a common physical layer. The method includes creating a first interprocess communication data structure associated with a first network address on a first network device. A first communication is established between the first network device and a second network device using the first interprocess communication data structure and the first network address. The first communication passes through the common physical layer for the first network device. A second interprocess communication data structure associated with a second network address is created on the first network device. The second network address is different from the first network address. A second communication is created between the first network device and a third network device using the second interprocess communication data structure and the second network address. The second communication also passes through the common physical layer for the first network device.

The method may help overcome limitations in having one network address common to every process on the host computer. With this method, each process may create its own interprocess communication data structure for communicating over a data network. In turn, each interprocess communication data structure may be associated with its own network address.

The foregoing and other features and advantages of preferred embodiments of the present invention will be more readily apparent from the following detailed description, which proceeds with references to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the present invention are described with reference to the following drawings, wherein:

FIG. 1 is a block diagram illustrating a network system;

5 FIG. 2 is a block diagram illustrating a protocol stack for a network device;

FIG. 3 is a block diagram illustrating a typical stack implementation;

FIG. 4 is a block diagram illustrating a stack implementation with multiple network addresses; and

FIG. 5 is a flow diagram illustrating a method for using multiple network addresses.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

FIG. 1 is a block diagram illustrating an exemplary data network 10 for an illustrative embodiment of the present invention. The data network 10 includes a backbone network 12 (e.g. the Internet), a first network device 14, and a second network device 16. The backbone network 12 may be public in the sense that it may be accessible by many users who may monitor communications on it. Additionally, although only one is illustrated, there may be multiple local area networks ("LANs") 20 coupled to the backbone network 12. Data packets may be transferred to/from the first network device 14 and the second network device 16 over the backbone network 12. For example, the devices may be assigned public network addresses on the Internet. A data channel between the first network device 14 and the second network device 16 may include routers or gateways (24, 26). However, other data network types and network devices can also be used and the present invention is not limited to the data network and network devices described for an illustrative embodiment.

For example, the routers (24, 26) may be edge routers. An edge router routes data packets between one or more networks such as a public network (e.g. backbone network 12) and a private network (e.g. LAN 20). Edge routers are commercially available from numerous sources, including those provided by 3Com Corporation of Santa Clara, California, Cisco Systems of San Jose, California, Lucent Technologies of Murray Hill, New Jersey, Lucent subsidiaries including Livingston Enterprises, Inc. of Pleasanton, California, and Ascend Communications of Alameda, California, and others.

In one exemplary preferred embodiment of the present invention, the first 14 and second 16 network devices are telephony devices or bulk data devices. Bulk data devices include Web-

the data network 10 with multiple data networks and/or multiple local area networks at various positions in the data network configuration 10 are possible.

An operating environment for network devices of the present invention includes a processing system with at least one high speed Central Processing Unit ("CPU") and a memory.

5 In accordance with the practices of persons skilled in the art of computer programming, the preferred embodiments are described below with reference to acts and symbolic representations of operations or instructions that are performed by the processing system, unless indicated otherwise. Such acts and operations or instructions are referred to as being "computer-executed," "CPU executed," or "executable."

10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
105
110
115
120
125
130
135
140
145
150
155
160
165
170
175
180
185
190
195
200
205
210
215
220
225
230
235
240
245
250
255
260
265
270
275
280
285
290
295
300
305
310
315
320
325
330
335
340
345
350
355
360
365
370
375
380
385
390
395
400
405
410
415
420
425
430
435
440
445
450
455
460
465
470
475
480
485
490
495
500
505
510
515
520
525
530
535
540
545
550
555
560
565
570
575
580
585
590
595
600
605
610
615
620
625
630
635
640
645
650
655
660
665
670
675
680
685
690
695
700
705
710
715
720
725
730
735
740
745
750
755
760
765
770
775
780
785
790
795
800
805
810
815
820
825
830
835
840
845
850
855
860
865
870
875
880
885
890
895
900
905
910
915
920
925
930
935
940
945
950
955
960
965
970
975
980
985
990
995
1000
1005
1010
1015
1020
1025
1030
1035
1040
1045
1050
1055
1060
1065
1070
1075
1080
1085
1090
1095
1100
1105
1110
1115
1120
1125
1130
1135
1140
1145
1150
1155
1160
1165
1170
1175
1180
1185
1190
1195
1200
1205
1210
1215
1220
1225
1230
1235
1240
1245
1250
1255
1260
1265
1270
1275
1280
1285
1290
1295
1300
1305
1310
1315
1320
1325
1330
1335
1340
1345
1350
1355
1360
1365
1370
1375
1380
1385
1390
1395
1400
1405
1410
1415
1420
1425
1430
1435
1440
1445
1450
1455
1460
1465
1470
1475
1480
1485
1490
1495
1500
1505
1510
1515
1520
1525
1530
1535
1540
1545
1550
1555
1560
1565
1570
1575
1580
1585
1590
1595
1600
1605
1610
1615
1620
1625
1630
1635
1640
1645
1650
1655
1660
1665
1670
1675
1680
1685
1690
1695
1700
1705
1710
1715
1720
1725
1730
1735
1740
1745
1750
1755
1760
1765
1770
1775
1780
1785
1790
1795
1800
1805
1810
1815
1820
1825
1830
1835
1840
1845
1850
1855
1860
1865
1870
1875
1880
1885
1890
1895
1900
1905
1910
1915
1920
1925
1930
1935
1940
1945
1950
1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
2020
2025
2030
2035
2040
2045
2050
2055
2060
2065
2070
2075
2080
2085
2090
2095
2100
2105
2110
2115
2120
2125
2130
2135
2140
2145
2150
2155
2160
2165
2170
2175
2180
2185
2190
2195
2200
2205
2210
2215
2220
2225
2230
2235
2240
2245
2250
2255
2260
2265
2270
2275
2280
2285
2290
2295
2300
2305
2310
2315
2320
2325
2330
2335
2340
2345
2350
2355
2360
2365
2370
2375
2380
2385
2390
2395
2400
2405
2410
2415
2420
2425
2430
2435
2440
2445
2450
2455
2460
2465
2470
2475
2480
2485
2490
2495
2500
2505
2510
2515
2520
2525
2530
2535
2540
2545
2550
2555
2560
2565
2570
2575
2580
2585
2590
2595
2600
2605
2610
2615
2620
2625
2630
2635
2640
2645
2650
2655
2660
2665
2670
2675
2680
2685
2690
2695
2700
2705
2710
2715
2720
2725
2730
2735
2740
2745
2750
2755
2760
2765
2770
2775
2780
2785
2790
2795
2800
2805
2810
2815
2820
2825
2830
2835
2840
2845
2850
2855
2860
2865
2870
2875
2880
2885
2890
2895
2900
2905
2910
2915
2920
2925
2930
2935
2940
2945
2950
2955
2960
2965
2970
2975
2980
2985
2990
2995
3000
3005
3010
3015
3020
3025
3030
3035
3040
3045
3050
3055
3060
3065
3070
3075
3080
3085
3090
3095
3100
3105
3110
3115
3120
3125
3130
3135
3140
3145
3150
3155
3160
3165
3170
3175
3180
3185
3190
3195
3200
3205
3210
3215
3220
3225
3230
3235
3240
3245
3250
3255
3260
3265
3270
3275
3280
3285
3290
3295
3300
3305
3310
3315
3320
3325
3330
3335
3340
3345
3350
3355
3360
3365
3370
3375
3380
3385
3390
3395
3400
3405
3410
3415
3420
3425
3430
3435
3440
3445
3450
3455
3460
3465
3470
3475
3480
3485
3490
3495
3500
3505
3510
3515
3520
3525
3530
3535
3540
3545
3550
3555
3560
3565
3570
3575
3580
3585
3590
3595
3600
3605
3610
3615
3620
3625
3630
3635
3640
3645
3650
3655
3660
3665
3670
3675
3680
3685
3690
3695
3700
3705
3710
3715
3720
3725
3730
3735
3740
3745
3750
3755
3760
3765
3770
3775
3780
3785
3790
3795
3800
3805
3810
3815
3820
3825
3830
3835
3840
3845
3850
3855
3860
3865
3870
3875
3880
3885
3890
3895
3900
3905
3910
3915
3920
3925
3930
3935
3940
3945
3950
3955
3960
3965
3970
3975
3980
3985
3990
3995
4000
4005
4010
4015
4020
4025
4030
4035
4040
4045
4050
4055
4060
4065
4070
4075
4080
4085
4090
4095
4100
4105
4110
4115
4120
4125
4130
4135
4140
4145
4150
4155
4160
4165
4170
4175
4180
4185
4190
4195
4200
4205
4210
4215
4220
4225
4230
4235
4240
4245
4250
4255
4260
4265
4270
4275
4280
4285
4290
4295
4300
4305
4310
4315
4320
4325
4330
4335
4340
4345
4350
4355
4360
4365
4370
4375
4380
4385
4390
4395
4400
4405
4410
4415
4420
4425
4430
4435
4440
4445
4450
4455
4460
4465
4470
4475
4480
4485
4490
4495
4500
4505
4510
4515
4520
4525
4530
4535
4540
4545
4550
4555
4560
4565
4570
4575
4580
4585
4590
4595
4600
4605
4610
4615
4620
4625
4630
4635
4640
4645
4650
4655
4660
4665
4670
4675
4680
4685
4690
4695
4700
4705
4710
4715
4720
4725
4730
4735
4740
4745
4750
4755
4760
4765
4770
4775
4780
4785
4790
4795
4800
4805
4810
4815
4820
4825
4830
4835
4840
4845
4850
4855
4860
4865
4870
4875
4880
4885
4890
4895
4900
4905
4910
4915
4920
4925
4930
4935
4940
4945
4950
4955
4960
4965
4970
4975
4980
4985
4990
4995
5000
5005
5010
5015
5020
5025
5030
5035
5040
5045
5050
5055
5060
5065
5070
5075
5080
5085
5090
5095
5100
5105
5110
5115
5120
5125
5130
5135
5140
5145
5150
5155
5160
5165
5170
5175
5180
5185
5190
5195
5200
5205
5210
5215
5220
5225
5230
5235
5240
5245
5250
5255
5260
5265
5270
5275
5280
5285
5290
5295
5300
5305
5310
5315
5320
5325
5330
5335
5340
5345
5350
5355
5360
5365
5370
5375
5380
5385
5390
5395
5400
5405
5410
5415
5420
5425
5430
5435
5440
5445
5450
5455
5460
5465
5470
5475
5480
5485
5490
5495
5500
5505
5510
5515
5520
5525
5530
5535
5540
5545
5550
5555
5560
5565
5570
5575
5580
5585
5590
5595
5600
5605
5610
5615
5620
5625
5630
5635
5640
5645
5650
5655
5660
5665
5670
5675
5680
5685
5690
5695
5700
5705
5710
5715
5720
5725
5730
5735
5740
5745
5750
5755
5760
5765
5770
5775
5780
5785
5790
5795
5800
5805
5810
5815
5820
5825
5830
5835
5840
5845
5850
5855
5860
5865
5870
5875
5880
5885
5890
5895
5900
5905
5910
5915
5920
5925
5930
5935
5940
5945
5950
5955
5960
5965
5970
5975
5980
5985
5990
5995
6000
6005
6010
6015
6020
6025
6030
6035
6040
6045
6050
6055
6060
6065
6070
6075
6080
6085
6090
6095
6100
6105
6110
6115
6120
6125
6130
6135
6140
6145
6150
6155
6160
6165
6170
6175
6180
6185
6190
6195
6200
6205
6210
6215
6220
6225
6230
6235
6240
6245
6250
6255
6260
6265
6270
6275
6280
6285
6290
6295
6300
6305
6310
6315
6320
6325
6330
6335
6340
6345
6350
6355
6360
6365
6370
6375
6380
6385
6390
6395
6400
6405
6410
6415
6420
6425
6430
6435
6440
6445
6450
6455
6460
6465
6470
6475
6480
6485
6490
6495
6500
6505
6510
6515
6520
6525
6530
6535
6540
6545
6550
6555
6560
6565
6570
6575
6580
6585
6590
6595
6600
6605
6610
6615
6620
6625
6630
6635
6640
6645
6650
6655
6660
6665
6670
6675
6680
6685
6690
6695
6700
6705
6710
6715
6720
6725
6730
6735
6740
6745
6750
6755
6760
6765
6770
6775
6780
6785
6790
6795
6800
6805
6810
6815
6820
6825
6830
6835
6840
6845
6850
6855
6860
6865
6870
6875
6880
6885
6890
6895
6900
6905
6910
6915
6920
6925
6930
6935
6940
6945
6950
6955
6960
6965
6970
6975
6980
6985
6990
6995
7000
7005
7010
7015
7020
7025
7030
7035
7040
7045
7050
7055
7060
7065
7070
7075
7080
7085
7090
7095
7100
7105
7110
7115
7120
7125
7130
7135
7140
7145
7150
7155
7160
7165
7170
7175
7180
7185
7190
7195
7200
7205
7210
7215
7220
7225
7230
7235
7240
7245
7250
7255
7260
7265
7270
7275
7280
7285
7290
7295
7300
7305
7310
7315
7320
7325
7330
7335
7340
7345
7350
7355
7360
7365
7370
7375
7380
7385
7390
7395
7400
7405
7410
7415
7420
7425
7430
7435
7440
7445
7450
7455
7460
7465
7470
7475
7480
7485
7490
7495
7500
7505
7510
7515
7520
7525
7530
7535
7540
7545
7550
7555
7560
7565
7570
7575
7580
7585
7590
7595
7600
7605
7610
7615
7620
7625
7630
7635
7640
7645
7650
7655
7660
7665
7670
7675
7680
7685
7690
7695
7700
7705
7710
7715
7720
7725
7730
7735
7740
7745
7750
7755
7760
7765
7770
7775
7780
7785
7790
7795
7800
7805
7810
7815
7820
7825
7830
7835
7840
7845
7850
7855
7860
7865
7870
7875
7880
7885
7890
7895
7900
7905
7910
7915
7920
7925
7930
7935
7940
7945
7950
7955
7960
7965
7970
7975
7980
7985
7990
7995
8000
8005
8010
8015
8020
8025
8030
8035
8040
8045
8050
8055
8060
8065
8070
8075
8080
8085
8090
8095
8100
8105
8110
8115
8120
8125
8130
8135
8140
8145
8150
8155
8160
8165
8170
8175
8180
8185
8190
8195
8200
8205
8210
8215
8220
8225
8230
8235
8240
8245
8250
8255
8260
8265
8270
8275
8280
8285
8290
8295
8300
8305
8310
8315
8320
8325
8330
8335
8340
8345
8350
8355
8360
8365
8370
8375
8380
8385
8390
8395
8400
8405
8410
8415
8420
8425
8430
8435
8440
8445
8450
8455
8460
8465
8470
8475
8480
8485
8490
8495
8500
8505
8510
8515
8520
8525
8530
8535
8540
8545
8550
8555
8560
8565
8570
8575
8580
8585
8590
8595
8600
8605
8610
8615
8620
8625
8630
8635
8640
8645
8650
8655
8660
8665
8670
8675
8680
8685
8690
8695
8700
8705
8710
8715
8720
8725
8730
8735
8740
8745
8750
8755
8760
8765
8770
8775
8780
8785
8790
8795
8800
8805
8810
8815
8820
8825
8830
8835
8840
8845
8850
8855
8860
8865
8870
8875
8880
8885
8890
8895
8900
8905
8910
8915
8920
8925
8930
8935
8940
8945
8950
8955
8960
8965
8970
8975
8980
8985
8990
8995
9000
9005
9010
9015
9020
9025
9030
9035
9040
9045
9050
9055
9060
9065
9070
9075
9080
9085
9090
9095
9100
9105
9110
9115
9120
9125
9130
9135
9140
9145
9150
9155
9160
9165
9170
9175
9180
9185
9190
9195
9200
9205
9210
9215
9220
9225
9230
9235
9240
9245
9250
9255
9260
9265
9270
9275
9280
9285
9290
9295
9300
9305
9310
9315
9320
9325
9330
9335
9340
9345
9350
9355
9360
9365
9370
9375
9380
9385
9390
9395
9400
9405
9410
9415
9420
9425
9430
9435
9440
9445
9450
9455
9460
9465
9470
9475
9480
9485
9490
9495
9500
9505
9510
9515
9520
9525
9530
9535
9540
9545
9550
9555
9560
9565
9570
9575
9580
9585
9590
9595
9600
9605
9610
9615
9620
9625
9630
9635
9640
9645
9650
9655
9660
9665
9670
9675
9680
9685
9690
9695
9700
9705
9710
9715
9720
9725
9730
9735
9740
9745
9750
9755
9760
9765
9770
9775
9780
9785
9790
9795
9800
9805
9810
9815
9820
9825
9830
9835
9840
9845
9850
9855
9860
9865
9870
9875
9880
9885
9890
9895
9900
9905
9910
9915
9920
9925
9930
9935
9940
9945
9950
9955
9960
9965
9970
9975
9980
9985
9990
9995
10000
10005
10010
10015
10020
10025
10030
10035
10040
10045
10050
10055
10060
10065
10070
10075
10080
10085
10090
10095
10100
10105
10110
10115
10120
10125
10130
10135
10140
10145
10150
10155
10160
10165
10170
10175
10180
10185
10190
10195
10200
10205
10210
10215
10220
10225
10230
10235
10240
10245
10250
10255
1

readable medium, which exist exclusively on the processing system or be distributed among multiple interconnected processing systems that may be local or remote to the processing system.

Network device protocol stack

FIG. 2 is a block diagram illustrating a protocol stack 50 for network devices, such as devices 14 and 16 in FIG. 1, in data network 10. The protocol stack 50 is typically executable code and data structures associated with a kernel for an operating system of the network device (14,16). The code resides in memory locations associated with the kernel and directs a CPU or CPUs to format and exchange network communications properly. The data structures are portions of memory that are used by the protocol stack code to retain dynamic and static values, and are also accessible and controllable by the kernel.

As is known in the art, the Open System Interconnection ("OSI") model may describe computer networks. The OSI model consists of seven layers including from lowest-to-highest, a physical, data-link, network, transport, session, presentation, and application layer. The physical layer exchanges bits over a communication link. The data link layer exchanges error free frames of data. The network layer exchanges and routes data packets.

The lowest layer of the protocol stack is the physical layer. The physical layer includes the physical media interfaces 52 that place signals on transmission media such as wires, coaxial cable, optical fiber, or transmit them as electromagnetic waves. The physical media interfaces 52 also read signals from the transmission media and present them to the data-link layer.

In the data-link layer is a Medium Access Control ("MAC") layer 54. As is known in the art, the MAC layer 54 controls access to a transmission medium via the physical layer. For more information on the MAC layer protocol 54 see IEEE 802.3 for Ethernet and IEEE 802.14 for

cable modems. However, other MAC layer protocols 54 could also be used and the present invention is not limited to IEEE 802.3 or IEEE 802.14.

Above the data-link layer is an Internet Protocol ("IP") layer 58. The IP layer 58, hereinafter IP 58, roughly corresponds to OSI layer 3, the network layer, but is typically not defined as part of the OSI model. As is known in the art, the IP 58 is a message addressing and delivery protocol designed to route traffic within a network or between networks. For more information on the IP 58 see IETF RFC-791, the contents of which are incorporated herein by reference.

The Internet Control Message Protocol ("ICMP") layer 56 is used for network management. The main functions of the ICMP layer 56, hereinafter ICMP 56, include error reporting, reachability testing (e.g., "pinging") congestion control, route-change notification, performance, subnet addressing and others. Since the IP 58 is an unacknowledged protocol, datagrams may be discarded and the ICMP 56 is used for error reporting. For more information on the ICMP 56 see IETF RFC-792, the contents of which are incorporated herein by reference.

Above the IP 58 and the ICMP 56 is a transport layer with a User Datagram Protocol layer 60 ("UDP"). The UDP layer 60, hereinafter UDP 60, roughly corresponds to OSI layer 4, the transport layer, but is typically not defined as part of the OSI model. As is known in the art, the UDP 60 provides a connectionless mode of communications with datagrams. For more information on the UDP 60 see IETF RFC-768, the contents of which are incorporated herein by reference. The transport layer may also include a connection-oriented Transmission Control Protocol ("TCP") layer 62. For more information on TCP see IETF RFC-793 and IETF RFC-1323, the contents of which are incorporated herein by reference.

Above the transport layer is an application layer where the application programs that carry out desired functionality for a network device reside. For example, the application programs for the network device 16 may include a printer application program, while application programs for the network device 14 may include a facsimile application program.

5 In the application layer are typically a Dynamic Host Configuration Protocol ("DHCP") layer 66 and a File Transfer Protocol ("FTP") layer 68. The DHCP layer 66 is a protocol for passing configuration information to and from hosts on an IP 58 network. For more information on the DHCP layer 66 see IETF RFC-2131, the contents of which are incorporated herein by reference. The FTP layer 68 is a file transfer protocol used to download files and configuration information. For more information on the FTP layer 68 see IETF RFC-959, the contents of which are incorporated herein by reference. Processes also reside in the application layer. While FIG. 2 identifies a five layer OSI model, those skilled in the art will recognize that more or fewer protocol layers may be used in the protocol stack 50.

Network address interfaces

FIG. 3 is a block diagram illustrating a typical stack implementation 80. The diagram depicts a typical TCP/IP or UDP/IP stack implementation in many operating systems. One or more processes 82-86 may each connect to one or more sockets 88-92. As is known to those skilled in the art, an application is a program that performs a useful task. An application may include many processes, each of which is a set of instructions for directing a CPU to perform a specific task.

20 Additionally, each process 82-86 may connect to one or more sockets 88-92 and each socket 88-92 may connect to one or more processes 82-86. In general, a process may connect to

more than one socket. Also, the processes within an application may simultaneously be associated with the same socket. For example: Process A 82 is connected to Socket #2 90; Process B 84 is connected to both Socket #1 88 and Socket #2 90; and Process C 86 is connected to Socket #2 90 and Socket #3 92.

5 Each socket includes a source IP 58 address as well as a source TCP 62 or UDP 60 port. Typically, each of the source IP 58 addresses in the sockets 88-92 are actually pointers to a host's single IP 58 address for its network address interface 94, which in turn is bound to a data-link interface 96. The IP 58 address typically resides in memory associated with a kernel of an operating system for the host network device. All Sockets 88-92 share the same IP 58 address and data-link layer, and communicate over the same physical medium.

As will be further described below, a function call may specify the IP 58 address for the socket, provided that this IP 58 address is a valid IP 58 address for the network address interface 94. More generally, however, the function call does not specify the IP 58 address. Instead, it allows the kernel of the operating system for the host network device to choose the (known) IP 58 address for the network address interface 94. Also, by calling the socket creation and binding functions, the process may specify a TCP 62 or UDP 60 port number (especially if the application uses a well-known port number as is familiar to those skilled in the art). Alternatively, the process may leave the choice of port up to the kernel of the operating system (the latter being referred to as an "ephemeral" port in the art).

20 An API for the sockets 88-92 includes reentrant functions for creating sockets and binding them to interfaces. As is known to those skilled in the art, a function is a self-contained set of instructions for carrying out a particular task. When a process calls a reentrant function,

the kernel of the operating system for the network device directs the CPU to execute the code of the function. After executing the code of the function, the kernel instructs the CPU to reenter the calling process, to return a result at the point where the calling process called the function, and to continue executing the code of the calling process. For more information on socket functions, see "The Design of the Unix Operating System", by Maurice J. Bach, Prentice-Hall, 1990, the contents of which are incorporated herein by reference. Generally, processes running on the host network device call the reentrant functions for the socket API.

In operation, a first process on a first network device 14, e.g. Process B 84 of FIG. 3, exchanges data with a second process (not shown) on a second network device 16 through an interprocess communication. An example of an interprocess communication is a socket-to-socket virtual data link. Data from the first process transfers to an appropriate interprocess communication data structure, e.g. Socket #1 88. The interprocess communication data structure is a portion of memory in the first network device 14 that is associated with the protocol stack 50 for the first network device 14. When the first process places data in this data structure, the kernel of the operating system for the first network device 14 encapsulates the data and transmits it as a packet to the second network device 16. Once the packet reaches the second network device 16, the kernel of the second network device 16 extracts the data and places it in another interprocess communication data structure. The other interprocess communication data structure is associated with the second process, e.g. a socket at the other end of the interprocess communication. The second process may access the other data structure to receive the data from the first process. The operating system creates and configures interprocess communications data structures through calls to an API.

API for a single network address

A call to a reentrant function called "socket" creates a socket by setting up a data structure in the host computer's memory. A "socket" call may take the following form:

`sd = socket (format, type, protocol)` (1)

- 5 The socket descriptor, "sd", is typically a handle that is used by the calling process to identify this particular socket. As is known in the art, a handle is code that includes access control information and/or a pointer to an object. For example, the socket descriptor appears in "read" and "write" statements and designates the input stream from which the host reads data or the output stream to which the host writes data.

10 The value of the "format" argument may indicate the address format of a communications domain. For example, as is known in the art, a value of AF_INET, a designator of the IP 58 address family, may indicate that the socket is to be used to communicate between network devices over an IP 58 connection. Additionally, the value of AF_INET indicates that the network devices have thirty-two bit addresses (i.e., corresponding to familiar IP 58 address dotted decimal notation w.x.y.z). Similarly, a value of AF_INET6 may indicate that communication is over IP version 6, which requires one-hundred twenty-eight bit addressing.

15 The values of the "type" and "protocol" arguments may indicate whether communication over the socket is connectionless or connection oriented. For example, as is known in the art, a value of SOCK_STREAM for the "type" argument may indicate that communication is a
20 connection oriented virtual circuit by means of TCP 62. Similarly, a value of SOCK_DGRAM for the "type" argument may indicate that communication is connectionless via datagrams by

means of UDP 60. As is known to those skilled in the art, the “protocol” argument may indicate a particular protocol to control the communication.

Once the kernel of the operating system creates the data structure for the socket, the socket may be bound to the network layer interface and assigned a network address and/or a port number. The “bind” reentrant function typically performs this task for servers and UDP 60 clients. A “bind” function call associates a name with the socket descriptor and is illustrated below:

bind (sd, host address, length) (2)

The socket descriptor, “sd,” is the value that was returned by the “socket” call referenced above (1) when the socket data structure was created. The “host address” argument is typically a pointer to a data structure stored in the kernel of the operating system for the host. The data structure typically designates an address family, e.g. AF_INET, an address for the host, and a port number for the socket. The value of the “length” argument is typically the size of the data structure referenced by the “host address” argument. For example, if the host network device is a server, the “host address” data structure may specify an IP 58 address and a port address provided these are valid addresses for the network address interface 94. Alternatively, the “host address” data structure may specify a well-known port number but not specify an associated IP 58 address. In this case, the kernel of the server would automatically bind the socket to the existing IP 58 address 94 of the network layer interface. Once server processes have bound addresses to sockets, they may announce the socket names to client processes, such as processes 82-86.

Alternatively, if the host network device is configured as a TCP 62 or UDP 60 client, both the values of the host IP 58 address and port number may remain unspecified. In this case, the kernel of the client may automatically bind the socket to the existing IP 58 address of the network address interface 94 and to an ephemeral port of a transport layer interface.

5 Instead of using the “bind” function, however, a TCP 62 or UDP 60 client may also bind a socket using a “connect” reentrant function. The “connect” reentrant function associates the local client’s socket with a target socket on the server. A “connect” call associates a target address with a local socket descriptor as illustrated below:

connect (sd, target address, length) (3)

10 The socket descriptor, “sd,” is the value that was returned when the client’s socket data structure was created by a call to a “socket” function. The “target address” argument is typically a pointer to another data structure stored in the host client. This “target address” data structure typically designates an IP 58 address for the server to which the client is connecting, and a port number for the target socket on the server. The value of the “length” argument is typically the size of the data structure referenced by the “host address” argument. The “connect” function call typically refers to the address structure for the server’s socket, not the client’s socket, and leaves the assignment of the host IP 58 address and port number for the client’s socket to client’s kernel.

15 Once a socket has been created with a “socket” function call and bound with a “bind” or “connect” function call, a process may determine a local network address and port number by a
20 call to a “getsockname” reentrant function. A “getsockname” function call associates an address structure with the socket descriptor and is illustrated below:

getsockname (sd, address, length) (4)

The socket descriptor, "sd," is the value that was returned when the socket data structure was created by a call to a "socket" function. This descriptor refers to the socket whose address we wish to determine. The "address" argument is typically a pointer to a data structure stored in the kernel of the operating system for the host. The kernel fills the "address" data structure with the IP 58 address and port number that the socket is currently using. The value of the "length" argument is typically the size of the data structure referenced by the "address" argument. The "getsockname" function is useful whenever other processes 82-86 need to know the actual values for the IP 58 address of the network address interface 94 and the port number of the transport interface. For example, when a server creates and binds a socket it may need to advertise the address of this socket to client network devices. A process on the server may get the socket's address and port number and pass this information on to the client. In this manner, the clients would have access to a target address for connecting to the server's socket.

When network communications no longer need a socket, a process typically de-allocates the socket by a "close" reentrant function. A "close" function call releases a connection for data communications over the network. Previous "connect" or "bind" function calls may have established the connection. The "close" function call is illustrated below:

close (sd) (5)

Both the socket descriptor and port number associated with the socket disappear from the protocol stack 50. The port number of the transport interface typically returns to a pool of port numbers. The network address for the network layer interface 94, however, does not necessarily disappear from the protocol stack 50 because it may be bound to a data-link interface.

Multiple network address interfaces

In the above description of the socket API, there was only one available network address interface 94 as depicted in FIG. 3. In new and emerging communication systems, such as those accomplishing Network Address Translation ("NAT") or Voice-over-IP ("VoIP"), different processes on the same host may be required to have different IP 58 addresses. NAT is described in IETF RFC 1631 and IETF RFC 2663, the contents of which are incorporated herein by reference. VoIP is described in ITU Recommendation H.323, the contents of which are incorporated herein by reference. As is known in the art, Recommendation H.323 defines negotiation and adaptation layers for video and audio over packet switched networks that do not offer guaranteed service or Quality of Service ("QoS").

FIG. 4 is a block diagram illustrating a stack implementation 100 with multiple network addresses. As is illustrated in FIG. 4, each socket in a multiple network address interface may use a different IP 58 address, as illustrated by IP 58 addresses X, Y, and Z. Additionally, all IP 58 addresses may be bound to the same data-link interface 96. For example, Socket #1 108 is bound to IP 58 address @X 114, Socket #2 110 is bound to IP 58 address @X 116, and Socket #3 112 is bound to IP 58 address @X 118. All the IP 58 interfaces 114-118 are bound to the same data-link interface 96. However, the present invention is not limited to the network and data-link interfaces as illustrated in FIG. 4 and other multiple address interfaces may alternatively be used.

In order to support multiple network addresses, as illustrated in FIG. 4, a host protocol stack 50 may require modification to support multiple IP 58 addresses in a single network layer. Furthermore, the protocol stack 50 may require modification to receive multiple IP 58 addresses

10 dynamically, through a mechanism such as DHCP 66. Alternatively, assignment of the multiple
network addresses may occur by selecting them from a pool of static IP 58 addresses.
Additionally, when an application requests a new network address, the protocol stack 50 may
request the network address from a network address server, receive the network address
5 assignment from the server, and associate that network address with the socket. It should be
understood that the present invention is not limited to these configurations for supporting
multiple network addresses and that many more configurations are possible but the foregoing is
preferred.

10 In accordance with a preferred embodiment, a new socket API may be constructed that
directs the kernel of the operating system to establish and use multiple network addresses in the
protocol stack 50. The new socket API behaves differently compared to the old socket API
described above; namely, that the new API permits multiple network addresses whereas the old
API only permits a single network address.

15 Although the corresponding functions in the new API are different in execution and
produce different results from those of the old API, they may be conceptually analogous to the
old socket API functions. A programming convention, known to those skilled in the art, is to
name new socket functions after analogous old socket functions. The naming convention is a
mnemonic device used by programmers for purposes of replacing old versions of functions with
new versions. As described below, the new API may contain a modified "socket" system call to
20 request that a particular socket allocate a new IP 58 address. Additionally, as further described
below, the new API may contain modified "bind," "connect," "getsockname," and "close"
functions.

Modified socket function

A modified "socket" reentrant function that accommodates multiple network addresses may be defined in a variety of ways. In one exemplary preferred embodiment of the modified "socket" function, the modified "socket" function includes an argument to direct the kernel to
5 assign a new network address to a newly created socket. For example, the function call may take the form:

`sd = socket (format, type, protocol, tuple)` (6)

where the value of the extra "tuple" argument may indicate to the kernel that this socket is to be assigned a new network address from a pool of available addresses (i.e. a static assignment) or
10 dynamically through some mechanism such as DHCP 66. A default value for "tuple" may indicate that the socket is to be assigned a common network address by the protocol stack 50 in a manner similar to the single address "socket" call referenced above (1).

Another exemplary preferred embodiment uses a new address family for the "format" argument of the socket function call to indicate that the kernel assigns a new network address to
15 the socket. For example, the modified "socket" function call may retain the form of Equation 1. However, a value such as AF_INET_TUPLE for the "format" argument may indicate to the kernel that it should assign a new IP 58 address to this socket. It should be understood that the present invention is not limited to these exemplary embodiments. Although not currently known
20 but within the capabilities of one skilled in the art, other ways of creating a modified socket with a new network address are possible.

Modified bind function

The new socket API may include a modified "bind" reentrant function that accommodates multiple network addresses. A socket that is created by the above-described modified "socket" function call, and has asked for a new network address, will be bound to the new network address by a call to the modified "bind" function. The form of the modified "bind" function may be analogous to the old "bind" function referenced above (2).

The modified "socket" function call has already indicated to the kernel that a new network address is associated with the socket descriptor. The modified "bind" function call passes the socket descriptor, "sd," to the kernel of the operating system for use in the protocol stack 50. The kernel recognizes this value for socket descriptor as associated with a new network address by a previous modified "socket" function call. As before, for the single network address interface, the calling process may specify a well-known port number as a parameter of the bind function call. Alternatively, the process may leave the port number unassigned, in which case the kernel may fill in an ephemeral port number as happened in the old "bind" function. The IP 58 address argument may typically be left unassigned in the "bind" function call.

The kernel of the operating system examines the memory associated with the protocol stack 50 and fills in the new IP 58 address that is associated with the socket descriptor, "sd." In one exemplary preferred embodiment, the kernel selects the IP 58 address from a pool of reserved addresses (i.e. static assignment). Alternatively, the kernel requests the address dynamically from an address server when the "socket" function call creates the socket. The IP 58 address may be stored in a table in memory associated with the protocol stack 50 section of the kernel, along with the socket descriptor. When a process later makes a modified "bind"

function call with the socket descriptor, the kernel searches for this socket descriptor. The kernel determines the previously reserved IP 58 address from the table, and returns this address in the IP 58 address structure of the modified "bind" function.

In another exemplary preferred embodiment, the kernel does not select the IP 58 address until a process calls the modified "bind" function. The call to the modified "socket" function records that the returned socket descriptor, sd, is slated to be assigned a new IP 58 address. The kernel, however, does not yet assign the network address. When the process makes the modified "bind" function call, the kernel determines that the socket descriptor argument contains a socket descriptor that was slated for a new IP 58 address. The kernel selects an IP 58 address for the socket from a pool of addresses or dynamically as described above. The kernel also associates the socket descriptor with the IP 58 address, e.g. in a table, and passes the new IP 58 address up to the IP 58 address structure in the modified "bind" function. It should be understood that the present invention is not limited to these exemplary embodiments and that many other ways of binding a modified socket with a new network address are possible.

Modified connect function

The "connect" reentrant function also requires modification to accommodate multiple network addresses. As described above, both TCP 62 and UDP 60 clients use a "connect" function call to allocate an IP 58 address and an ephemeral port. The modified "connect" function may retain the form of the old "connect" function call referenced above (3). The target IP 58 address and port number (the address of the server associated with the client) are passed as arguments to the modified "connect" function as before.

The modified "connect" function call passes the socket descriptor, "sd," to the kernel. The kernel recognizes this value for the socket descriptor as being associated with a new network address by a previous modified "socket" function call. As before, for the single network address interface 94, the calling process may allow the kernel to bind an ephemeral port number to the
5 socket.

When a process makes a "connect" function call, the kernel also binds the socket to the new IP 58 address that is associated with the socket descriptor, "sd." In one exemplary preferred embodiment, the kernel selects the IP 58 address from a pool of reserved addresses (i.e. a static assignment). Alternatively, the process may request the address dynamically from an address
10 server (not shown in FIG. 4) when the "socket" function call creates the socket. The IP 58 address may be stored in a table in memory associated with the protocol stack 50 section of the kernel, along with the socket descriptor. A later modified "connect" function call with the socket descriptor causes the kernel to search for this socket descriptor and determine the previously
15 reserved IP 58 address from the table. In this manner, a connection is established between the client socket, having the new IP 58 address and port number, and the server socket, having the target IP 58 address and port number.

In another exemplary preferred embodiment, the kernel does not assign an IP 58 address until the call is made to the modified "connect" function. The call to the modified "socket" function records that the returned socket descriptor, sd, is slated to be assigned a new IP 58
20 address. The kernel, however, does not yet assign the network address. When a process makes the modified "connect" function call, the kernel determines that the socket descriptor argument contains a socket descriptor that slated for a new IP 58 address. The kernel may select an IP 58

address for the socket from a pool of addresses or, alternatively, the IP 58 address may be assigned dynamically such as is described above. The kernel also associates the socket descriptor with the IP 58 address, e.g. in a table. In this manner, a connection is established between the client socket, having the new IP 58 address and port number, and the server socket, having the target IP 58 address and port number. It should be understood that the present invention is not limited to these exemplary embodiments and that many other ways of connecting a modified socket with a new network address are possible.

Modified getsockname function

The "getsockname" reentrant function also requires modification to accommodate multiple network addresses. The modified "getsockname" function may retain the form of the old "getsockname" function as referenced above (4). Although the modified "getsockname" function has the same name as the old "getsockname" function, its operation is different. A process calls the modified "getsockname" function and passes the value of the socket descriptor to the kernel. The kernel determines whether the socket descriptor is associated with a new IP 58 address. If the socket descriptor is associated with a new IP 58 address, the kernel fills in the referenced address structure with the new IP 58 address and port number. In this manner, the modified "getsockname" function returns the new network address assigned to the socket in an address data structure. The new network address may have been associated with the socket descriptor during a previous modified "socket" function call, a previous modified "bind" function call, or a previous modified "connect" function call.

Modified close function

The "close" reentrant function is preferably also modified to accommodate multiple network addresses. The modified "close" function may retain the form of the old "close" function as referenced above (5). A process calls the "close" function and passes the value of the socket descriptor to the kernel. The kernel determines whether this socket descriptor is associated with a new IP 58 address. The new network address may have been associated with the socket descriptor during a previous modified "socket" function call, a previous modified "bind" function call, or a previous modified "connect" function call. If the socket descriptor is associated with the new IP 58 address, the kernel also de-allocates this new IP 58 address and port number. The kernel accomplishes this, for example, by deleting the socket descriptor and/or the new network address from a table in the protocol stack 50 section of the kernel. In this manner, the modified "close" function releases the data communications connection, and the network and virtual interface both disappear. Alternatively, if the kernel assigned the network address to the socket through a dynamic process such as DHCP 66, the socket may automatically close at the end of a lease time by methods known to those skilled in the art.

Method for using multiple network addresses

FIG. 5 is a flow diagram illustrating a method 130 for using multiple network addresses for interprocess communication through a common physical layer. The method 130 includes creating a first interprocess communication data structure associated with a first network address on a first network device 14 at step 132. The first network device 14 may be the telephony device as illustrated in FIG. 1, although other types of network devices are possible and the present invention is not limited to the telephony device of FIG. 1. An example of an interprocess

communication data structure associated with a network address is a socket that is assigned a new IP 58 address when it is created, as described above. A call to a "socket" function from a process may create the first socket as a data structure in memory associated with the kernel of the first network device 14. For example Socket #1 108 of FIG. 4 may be created by a modified
5 "socket" function call from Process B 84. However, it should be understood that the present invention is not limited to sockets and IP 58 addresses, and that other types of interprocess communication data structures and network addresses may be used.

At step 134, a first communication is established between the first network device 14 and a second network device 16 using the first interprocess communication data structure and the first network address. For example, the first socket may establish a connection with a target socket on the second network device 16. This may initiate a data flow between the process bound to the first socket and another process on the second network device 16 bound to the target socket. The communication may be connection oriented, such as a TCP 62 virtual connection, or connectionless, such as a UDP 60 virtual connection. The data flow may follow a modified
10 "bind" or "connect" function call with the "socket descriptor" for the first socket. Process B 84, for example, may initiate a data flow through the data structure of Socket #1 108. The kernel associated Socket #1 108 with its own network interface 114 having the new IP 58 address @X. However, it should be understood that the present invention is not limited to TCP/IP communications or the like and that other forms of communications are possible, such as UDP/IP
15 or X.25 communications familiar to those skilled in the art.

The first communication passes through the common physical layer for the first network device 14. As illustrated in FIG. 4, the multiple IP 58 addresses at the network layer share a

common data-link and device driver 96 for the first network device 14 at its physical layer. Data flows to and from all processes 82-86 through the same physical layer interface 96. Data to or from one process 82-86 may go to one or more socket data structures 108-112. The respective network layer interface 114 for the first socket 108 encapsulates or decapsulates the data and processes the packets through the common data-link and physical layer 96.

A second interprocess communication data structure is created at step 136. The second interprocess communication data structure is associated with a second network address on the first network device. The second network address is different from the first network address. For example, Process B 84, having already created Socket #1 108, may also create a second socket, Socket #2 110, by another modified "socket" function call. Socket #2 110 has a different IP 58 address, @Y, compared to Socket #1 108, @X, as the kernel assigns a new IP 58 address when the process makes a modified "socket" function call. There are now two sockets on the first network device 14, and each socket is associated with a different IP 58 address.

At step 138, a second communication is established between the first network device 14 and a third network device (not shown) using the second interprocess communication data structure and the second network address. The second socket may then establish a connection with another target socket on a third computer to provide another virtual connection. This may initiate a data flow between the process and a third process on the third network device bound to the target socket. The communication may be connection oriented, such as a TCP 62 virtual connection, or connectionless, such as a UDP 60 virtual connection. The data flow may follow a modified "bind" or "connect" function call with the "socket descriptor" for the second socket. Process B 84, for example, may initiate a data flow through the data structure of Socket #2 110.

The kernel associated Socket #2 110 with its own network interface 116 having the new IP 58 address @Y. The respective network layer interface 116 for the second socket 110 encapsulates or decapsulates the data and processes the packets through the common data-link and physical layer 96. In this manner, a host computer, application, process, or other entity may support multiple network addresses and bind each address to a separate socket and/or a separate process. This allows a network device to communicate with other network devices using two or more network addresses.

In the present invention, more than one IP 58 interface may map to the same physical or logical data-link device. However, each interface will only send and receive data on behalf of a related group of processes as illustrated in FIG. 4. An advantage of the present invention compared to a traditional data-link interface is that it represents an IP 58 address that is bound to a given executing instance of an application (e.g., a process or related group of processes). Such a configuration may be useful for differentiating IP 58 data traffic to or from a particular host based on something other than a transport-layer parameter such as a TCP 62 or UDP 60 port.

For example, in Internet telephony it may be advantageous for each new call to be ascribed a new IP 58 address. Calls are typically IP 58 messages exchanged between telephony devices in a virtual private network. The telephony devices may typically support multiple calls, each call being associated with a process in the telephony device. In this case, the new IP 58 address resides in a private network address space for the virtual private network. NAT tunnels the calls across the (public) Internet. Ascribing a new and unique IP 58 address to each call may ensure that the identity of the caller is hidden as the call traverses the Internet. Each call process

should be associated with a respective private IP 58 address. The present invention may assign a new IP 58 address to each new call in the manner described above.

Another advantage of supporting multiple IP 58 addresses is improving switching efficiency. As described above, IP 58 addresses rather than TCP 62 or UDP 60 port number may distinguish traffic for applications. In general, layer 3 (network layer) switching using IP 58 addresses is expected to be more efficient than layer 4 (transport layer) switching using TCP 62 or UDP 60 ports. Improved switching efficiency may increase the call capacity of the Internet.

Yet another advantage of using multiple IP 58 addresses is to differentiate data traffic associated with different Quality of Service ("QoS"). QoS provides statistical guarantees of throughput, delay, delay variation, and packet loss. QoS is important in the transmission of Internet telephony, multimedia, and video data streams as these transmissions require a guaranteed bandwidth to function. Different applications on the same host computer may need to communicate with a separate QoS. For example, Internet telephony applications may require a constant or guaranteed bitrate QoS, whereas a web browsing application may require a basic QoS. Differentiating QoS may also improve the ability of the Internet to carry many forms of data communication simultaneously.

Edge routers (24,26) supporting QoS are required to process IP 58 packets differently depending on the QoS of the data communication. Instead of examining the internal details of an incoming IP 58 packet to determine the appropriate QoS, the edge router (24,26) may simply recognize that an IP 58 address in the header is associated with a particular QoS and process that packet accordingly. As discussed above, layer 3 switching is typically expected to be efficient. Ascribing an IP 58 address to each application may allow the IP 58 address for the application to

be associated with the QoS of the application. The IP 58 address and QoS of the application may be conveyed to the edge router (24,26) to establish a layer 3 switching channel for the application. Therefore, each communication is over a virtual channel between applications associated with a particular QoS and a unique IP 58 address.

5 It should be understood that the programs, processes, methods, systems and apparatus described herein are not related or limited to any particular type of computer apparatus (hardware or software), unless indicated otherwise. Various types of general purpose or specialized computer apparatus may be used with or perform operations in accordance with the teachings described herein.

10 In view of the wide variety of embodiments to which the principles of the invention can be applied, it should be understood that the illustrated embodiments are exemplary only, and should not be taken as limiting the scope of the present invention. For example, the steps of the flow diagrams may be taken in sequences other than those described, and more or fewer elements or components may be used in the block diagrams.

15 The claims should not be read as limited to the described order or elements unless stated to that effect. In addition, use of the term "means" in any claim is intended to invoke 35 U.S.C. §112, paragraph 6, and any claim without the word "means" is not so intended. Therefore, all implementations that come within the scope and spirit of the following claims and equivalents thereto are claimed as the invention.

WE CLAIM:

1. A method for using multiple network addresses for interprocess communication through a common physical layer, comprising:

creating a first interprocess communication data structure associated with a first network address on a first network device;

5 establishing a first communication between the first network device and a second network device using the first interprocess communication data structure and the first network address, wherein the first communication passes through the common physical layer for the first network device;

creating a second interprocess communication data structure associated with a second network address on the first network device, wherein the second network address is different from the first network address; and

10 establishing a second communication between the first network device and a third network device using the second interprocess communication data structure and the second network address, wherein the second communication passes through the common physical layer for the first network device.

2. A computer readable medium having stored therein instructions for causing a central processing unit to execute the method of Claim 1.

3. The method of Claim 1 wherein the first interprocess communication data
5 structure is a first socket comprising:

a first socket descriptor with which a first process on the first network device accesses the
first interprocess communication data structure; and
the first network address.

4. The method of Claim 1 wherein the second interprocess communication data
structure is a second socket comprising:

a second socket descriptor with which a second process on the first network device
accesses the second interprocess communication data structure; and
5 the second network address.

5. The method of Claim 1 wherein the first network address and the second network
address are Internet Protocol addresses.

6. The method of Claim 1 wherein the step of creating the first or second
interprocess communication data structure includes calling a reentrant socket networking
function that allows multiple network addresses to be allocated.

7. The method of Claim 1 wherein the step of creating the first or second
interprocess communication data structure includes calling a reentrant bind socket networking
function that allows multiple network addresses to be allocated.

8. The method of Claim 1 wherein the step of establishing the first or second communication includes calling a reentrant connect socket networking function that allows multiple network addresses to be allocated.

9. A computer readable medium having stored therein a library of reentrant functions generally available to, and callable by, a plurality of client applications on a host computer, the computer readable medium comprising:

a first reentrant networking function for creating a socket that is associated with a new network address, wherein the first function allocates a memory structure in the host computer for the socket and creates a new network layer interface for the new network address, and wherein the first reentrant networking function allows multiple network addresses to be used with a common physical layer;

a second reentrant networking function for binding the socket to the new network address, wherein the second reentrant networking function allows multiple network addresses to be used with the common physical layer;

a third reentrant networking function for connecting the socket to a target socket on another host computer, wherein the third function binds the socket to the new network address, wherein the third reentrant networking function allows multiple network addresses to be used with the common physical layer;

a fourth reentrant networking function for determining the new network address associated with the socket, wherein the fourth reentrant networking function allows multiple network addresses to be used with the common physical layer; and

a fifth reentrant networking function for closing the socket, wherein the fifth reentrant
20 networking function allows multiple network addresses to be used with the common physical
layer.

10. A computer readable medium having stored therein an application programming
interface having a plurality of function interfaces for networking functions stored in an external
library generally available to, and callable by, a plurality of client applications on a host
computer, the computer readable medium comprising:

5 a first function interface for calling a first networking function for creating a socket that is
associated with a new network address, wherein the first networking function allocates a memory
structure in the host computer for the socket and creates a new network layer interface for the
new network address, and wherein the first function interface allows multiple network addresses
to be used over a common physical layer;

10 a second function interface for calling a second networking function for binding the
socket to the new network address, wherein the second function interface allows multiple
network addresses to be used over the common physical layer;

a third function interface for calling a third networking function for connecting the socket
to a target socket on another host computer, wherein the third function binds the socket to the
15 new network address, and wherein the third function interface allows multiple network addresses
to be used over the common physical layer;

a fourth function interface for calling a fourth networking function for determining the new network address associated with the socket, wherein the fourth function interface allows multiple network addresses to be used over the common physical layer; and

20 a fifth function interface for calling a fifth reentrant networking function for closing the socket, wherein the fifth function interface allows multiple network addresses to be used over the common physical layer.

11. A computer readable medium having stored therein a reentrant networking function for creating a socket, generally available to, and callable by, a plurality of applications on a host computer, comprising:

associating a descriptor with the socket, wherein the descriptor identifies the socket to
5 other reentrant network functions, and wherein the socket is to be assigned a new network address;

storing the descriptor in a protocol stack for the host computer; and

instructing the host computer to allocate a memory structure for the socket.

12. The computer readable medium of Claim 11 wherein the storing step comprises:

ascertaining the new network address in the protocol stack for the host computer;

associating the descriptor with the new network address; and

5 storing the descriptor and the new network address in the protocol stack of the host computer.

13. The computer readable medium of Claim 12 wherein the ascertaining step comprises:

requesting a dynamically assigned network address from an address server; and
receiving the new network address from the address server in response.

5

14. The computer readable medium of Claim 12 wherein the ascertaining step comprises:

selecting the new network address from a pool of network addresses on the host computer.

15. The computer readable medium of Claim 11 wherein the instructing step further comprises:

returning a value for the descriptor to an application that has called the reentrant networking function.

16. The computer readable medium of Claim 11 wherein the new network address is an Internet Protocol address.

17. A computer readable medium having stored therein a reentrant networking function for binding a socket, generally available to, and callable by, a plurality of applications on a host computer, comprising:

identifying the socket from a descriptor that is passed from an application that has called
5 the reentrant networking function;

determining that the descriptor is associated with a new network address in a protocol stack of the host computer; and

associating the new network address with the socket.

18. The computer readable medium of Claim 17 wherein the identifying step comprises:

matching the descriptor to a stored descriptor value in the protocol stack, wherein the stored descriptor value is associated with the socket.

19. The computer readable medium of Claim 17 wherein the determining step comprises:

matching the descriptor to a stored descriptor value in the protocol stack, wherein the stored descriptor value is associated with a stored network address; and

5 equating the new network address with the stored network address.

20. The computer readable medium of Claim 17 wherein the determining step comprises:

ascertaining the new network address in the protocol stack for the host computer;

associating the descriptor with the new network address; and

5 storing the descriptor and the new network address in the protocol stack of the host computer.

21. The computer readable medium of Claim 20 wherein the ascertaining step comprises:

requesting a dynamically assigned network address from an address server; and

receiving the new network address from the address server in response.

22. The computer readable medium of Claim 20 wherein the ascertaining step comprises:

selecting the new network address from a pool of network addresses on the host computer.

23. The computer readable medium of Claim 20 wherein the associating step further comprises:

returning a value for the new network address to the application that has called the reentrant networking function.

24. The computer readable medium of Claim 17 wherein the new network address is an Internet Protocol address.

25. A computer readable medium having stored therein a reentrant networking function for connecting a host socket to a target socket, generally available to, and callable by, a plurality of applications on a host computer, comprising:

identifying the host socket from a descriptor that is passed from an application that has
5 called the reentrant networking function;

determining that the descriptor is associated with a new network address in a protocol stack of the host computer;

associating the new network address with the host socket; and

allocating a memory structure for communication between the host socket and the target
10 socket.

26. The computer readable medium of Claim 25 wherein the identifying step comprises:

matching the descriptor to a stored descriptor value in the protocol stack, wherein the stored descriptor value is associated with the host socket.

27. The computer readable medium of Claim 25 wherein the determining step comprises:

matching the descriptor to a stored descriptor value in the protocol stack, wherein the stored descriptor value is associated with a stored network address; and

5 equating the new network address with the stored network address.

28. The computer readable medium of Claim 25 wherein the determining step comprises:

ascertaining the new network address in the protocol stack for the host computer;

associating the descriptor with the new network address; and

5 storing the descriptor and the new network address in the protocol stack of the host computer.

29. The computer readable medium of Claim 28 wherein the ascertaining step comprises:

requesting a dynamically assigned network address from an address server; and

receiving the new network address from the address server in response.

5

30. The computer readable medium of Claim 28 wherein the ascertaining step comprises:

selecting the new network address from a pool of network addresses on the host computer.

31. The computer readable medium of Claim 25 wherein the new network address is an Internet Protocol address.

32. A computer readable medium having stored therein a reentrant networking function for determining a new network address associated with a socket, generally available to, and callable by, a plurality of applications on a host computer, comprising:

identifying the socket from a descriptor that is passed from an application that has called
5 the reentrant networking function;

determining that the descriptor is associated with a new network address in a protocol stack of the host computer; and

returning a value for the new network address to the application that has called the reentrant networking function.

33. The computer readable medium of Claim 32 wherein the identifying step comprises:

matching the descriptor to a stored descriptor value in the protocol stack, wherein the stored descriptor value is associated with the socket.

34. The computer readable medium of Claim 32 wherein the determining step comprises:

matching the descriptor to a stored descriptor value in the protocol stack, wherein the stored descriptor value is associated with a stored network address; and

5 equating the new network address with the stored network address.

35. The computer readable medium of Claim 32 wherein the new network address is an Internet Protocol address.

36. A computer readable medium having stored therein a reentrant networking function for closing a socket, generally available to, and callable by, a plurality of applications on a host computer, comprising:

identifying the socket from a descriptor that is passed from an application that has called the reentrant networking function;

determining that the descriptor is associated with a new network address in a protocol stack of the host computer; and

de-allocating the new network address.

37. The computer readable medium of Claim 36 wherein the de-allocating step comprises:

matching the descriptor to a stored descriptor value in the protocol stack, wherein the stored descriptor value is associated with the new network address; and

5 deleting the stored descriptor value from the protocol stack.

38. The computer readable medium of Claim 37 further comprising:

deleting the new network address from the protocol stack.

39. The computer readable medium of Claim 36 wherein the new network address is an Internet Protocol address.

ABSTRACT

A method and application programming interface for using multiple network addresses on a common physical layer. The host protocol stack supports multiple Internet Protocol interfaces. When a process makes a function call to create a new socket, a new IP address is associated with the socket. Each socket is then bound to an IP address that is distinct from the IP addresses bound to other sockets. This is in contrast to conventional sockets that are bound to a common IP address. In this manner, each process may be associated with a unique IP address. Such a configuration may useful in Internet telephony where each call process receives a unique private IP address in a virtual private network.

FIG. 1

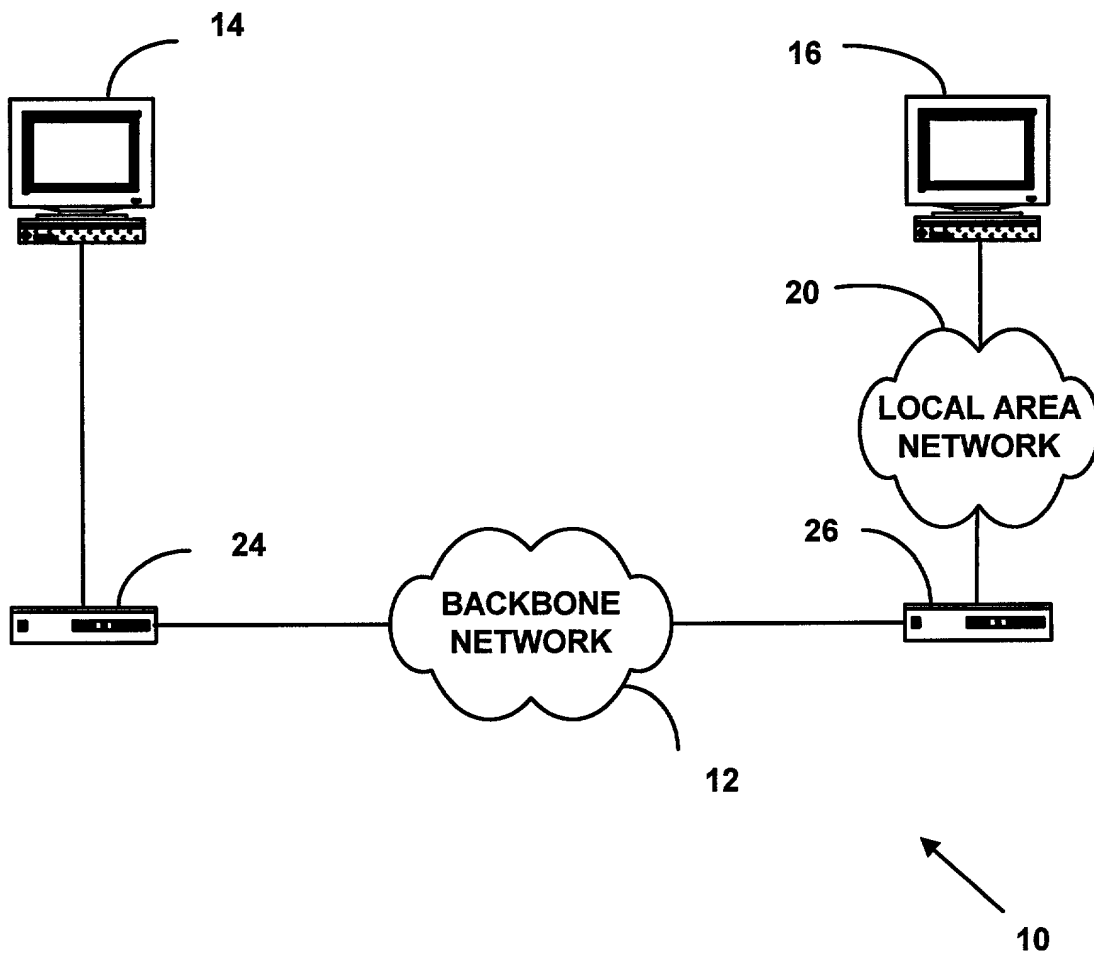


FIG. 2

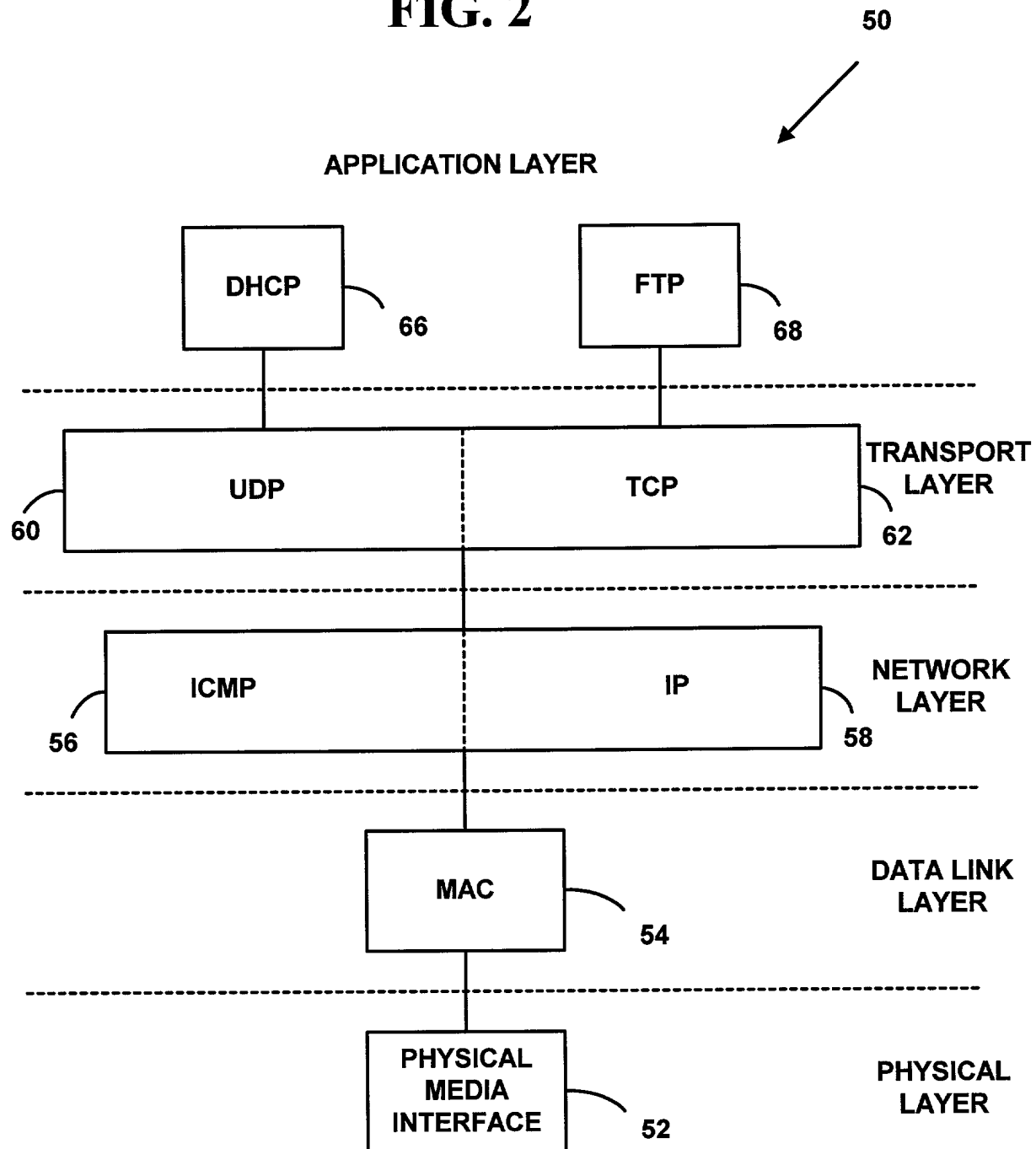


FIG. 3

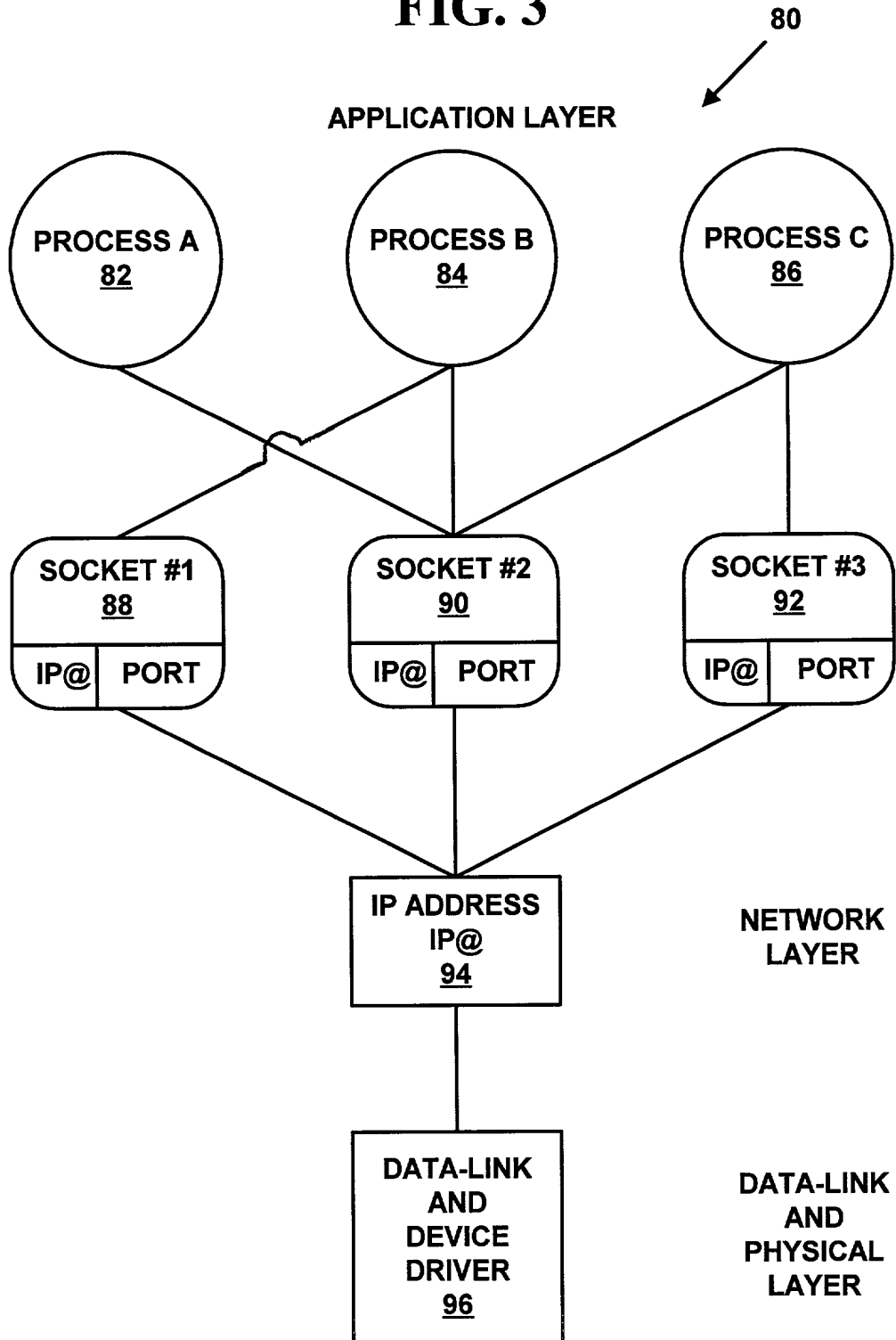


FIG. 4

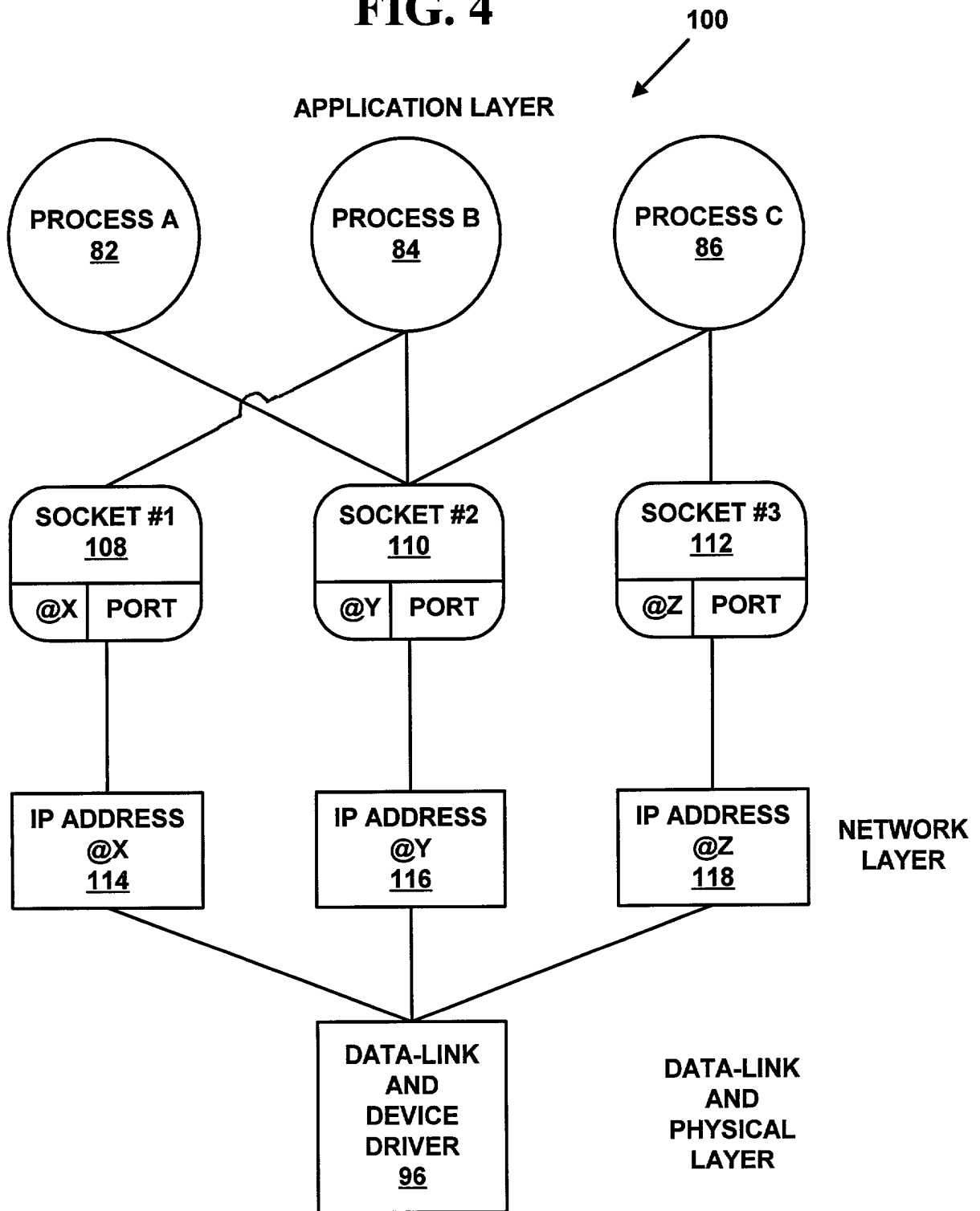
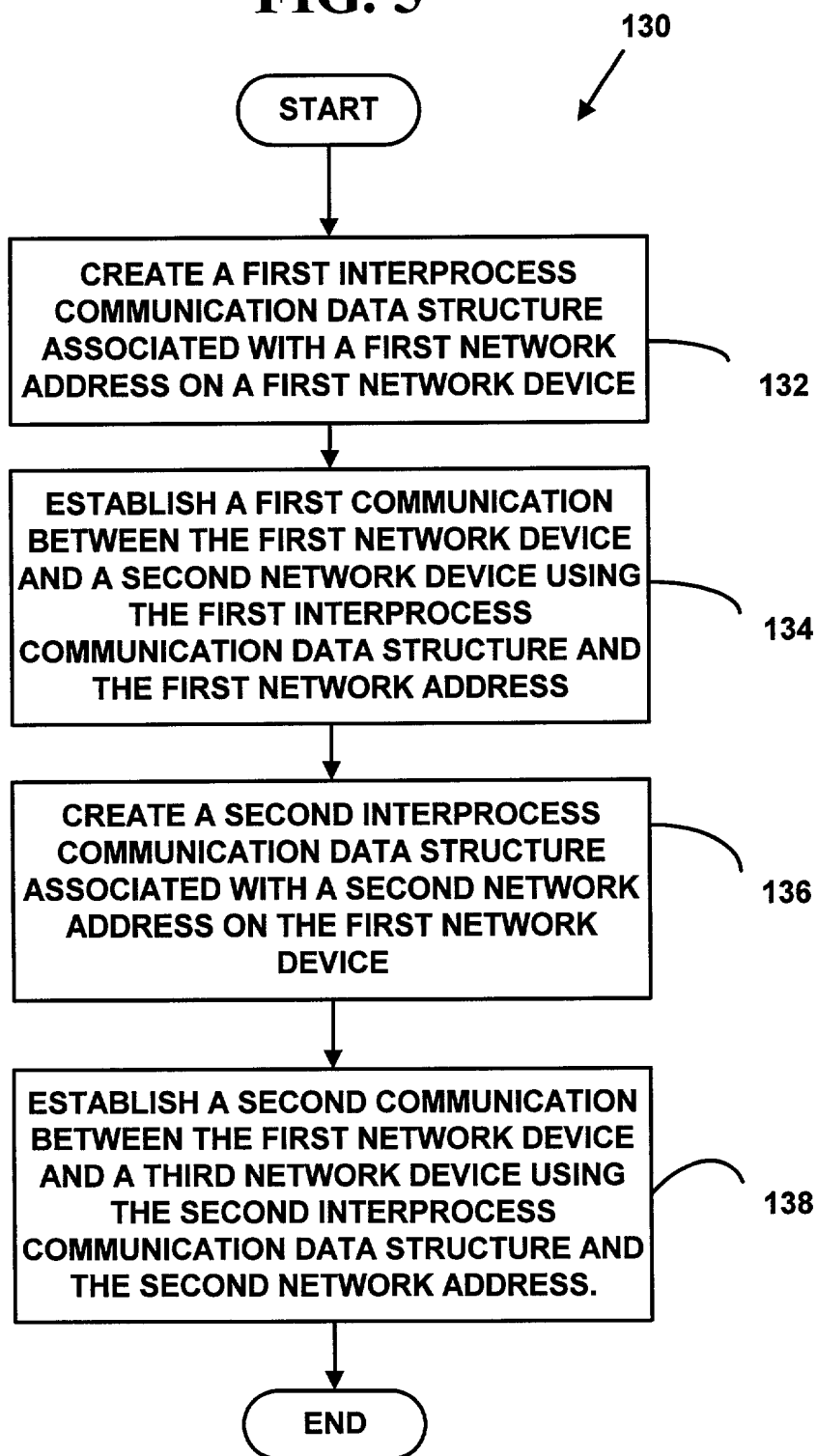


FIG. 5



**DECLARATION AND POWER OF ATTORNEY
FOR PATENT APPLICATION**

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name.

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

**METHOD AND APPLICATION PROGRAMMING INTERFACE
FOR ASSIGNING MULTIPLE NETWORK ADDRESSES**

the specification of which is attached hereto unless the following space is checked:

☐ was filed on _____ as United States Application Serial Number _____

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in 37 CFR § 1.56.

I hereby claim foreign priority benefits under 35 U.S.C. § 119(a)-(d) or § 365(b) of any foreign application(s) for patent or inventor's certificate, or § 365(a) of any PCT international application which designated at least one country other than the United States, listed below and have also identified below, by checking the box, any foreign application for patent or inventor's certificate, or PCT international application having a filing date before that of the application on which priority is claimed.

Prior Foreign Application(s):

	<u>Number</u>	<u>Country</u>	<u>Day/Month/Year Filed</u>
1.			
2.			

I hereby claim the benefit under 35 U.S.C. § 119(e) of any United States provisional application(s) listed below:

	<u>Application Number</u>	<u>Filing Date</u>
1.		
2.		

I hereby claim the benefit under 35 U.S.C. § 120 of any United States application(s), or § 365(c) of any PCT international application designating the United States, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT international application in the manner provided by the first paragraph of 35 U.S.C. § 112, I acknowledge the duty to disclose information which is material to patentability as defined in 37 C.F.R. § 1.56 which became available between the filing date of the prior application and the national or PCT international filing date of this application.

Application Number

Filing Date

Status: patented, pending, abandoned

1.

2.

I hereby appoint the following attorneys and agent(s) to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith:

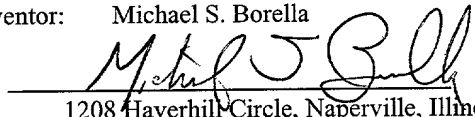
Denis A. Berntsen	Reg. No. 26707	Anthoula Pomrening	Reg. No. 38805
John J. McDonnell	Reg. No. 26949	George I. Lee.	Reg. No. 39269
Daniel A. Boehnen	Reg. No. 28399	James M. McCarthy	Reg. No. 39296
Bradley J. Hulbert	Reg. No. 30130	Jeremy Noe (agent)	Reg. No. 40104
Paul H. Berghoff	Reg. No. 30243	Sean M. Sullivan	Reg. No. 40191
Grantland G. Drutchas	Reg. No. 32565	Timothy R. Baumann	Reg. No. 40502
Steven J. Sarussi	Reg. No. 32784	Amir N. Penn	Reg. No. 40767
David M. Frischkorn	Reg. No. 32833	Patrick J. Halloran	Reg. No. 41053
James C. Gumina	Reg. No. 32898	Joshua R. Rich	Reg. No. 41269
A. Blair Hughes	Reg. No. 32901	Thomas E. Wettermann	Reg. No. 41523
Thomas A. Fairhall	Reg. No. 34591	Vernon W. Francissen	Reg. No. 41762
Emily Miao	Reg. No. 35285	Robert J. Irvine	Reg. No. 41865
Kevin E. Noonan	Reg. No. 35303	Richard A. Machonkin	Reg. No. 41962
Leif R. Sigmond, Jr.	Reg. No. 35680	David S. Harper	Reg. No. 42636
Lawrence H. Aaronson	Reg. No. 35818	Stephen Lesavich	Reg. No. 43749
Matthew J. Sampson	Reg. No. 35999	Enrique Perez	Reg. No. 43853
Curt J. Whitenack	Reg. No. 36054	Marcus J. Thymian	Reg. No. 43954
Christopher M. Cavan	Reg. No. 36475	S. Richard Carden	Reg. No. 44588
Michael S. Greenfield	Reg. No. 37142	Mark Chael	Reg. No. 44601
Roger P. Zimmerman	Reg. No. 38670	Stephen H. Docter	Reg. No. 44659

Address all telephone calls to Matthew J. Sampson at (312) 913-0001.

Address all correspondence to MCDONNELL BOEHNEN HULBERT & BERGHOFF, 300 South Wacker Drive, Chicago, Illinois 60606 USA.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full name of first inventor: Michael S. Borella

Inventor's signature: 

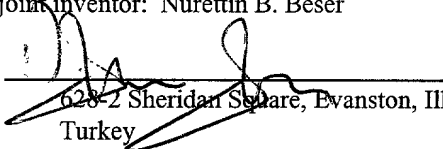
Date: 2/22/2000

Residence: 1208 Haverhill Circle, Naperville, Illinois 60563

Citizenship: United States of America

Post Office Address: 1208 Haverhill Circle, Naperville, Illinois 60563

Full name of second joint inventor: Nurettin B. Beser

Inventor's signature: 

Date: 22 Feb '00

Residence: 628-2 Sheridan Square, Evanston, Illinois 60202

Citizenship: Turkey

Post Office Address: 628-2 Sheridan Square, Evanston, Illinois 60202